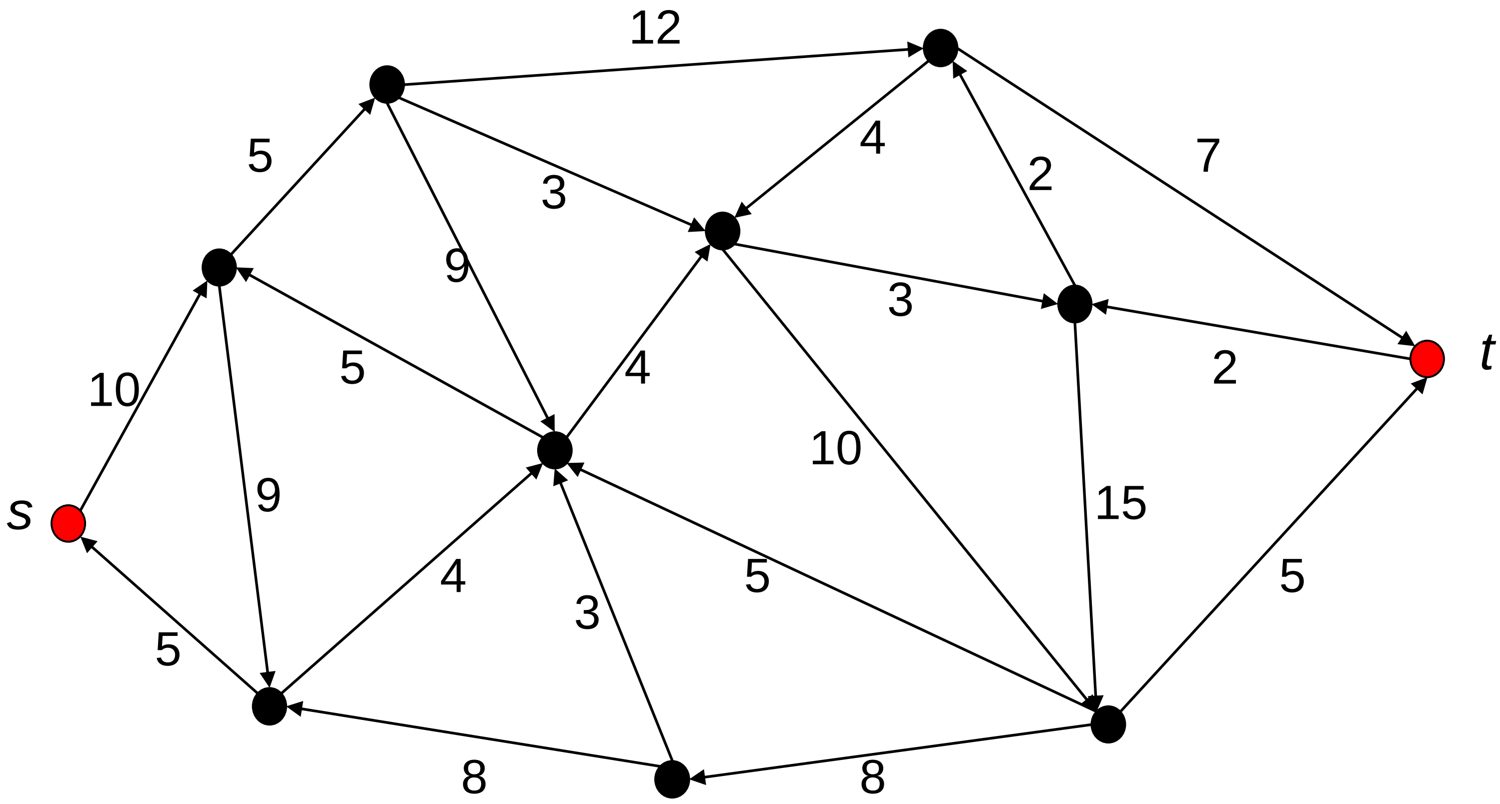
14 – Network Flow

1. Maximum flow problem



Maximum flow problem

$N = (V,E,c)$ directed network

$G = (V,E)$ directed graph, $c: E \to \mathbb{R}^+$ edge capacities

$s,t \in V$ source s, sink t

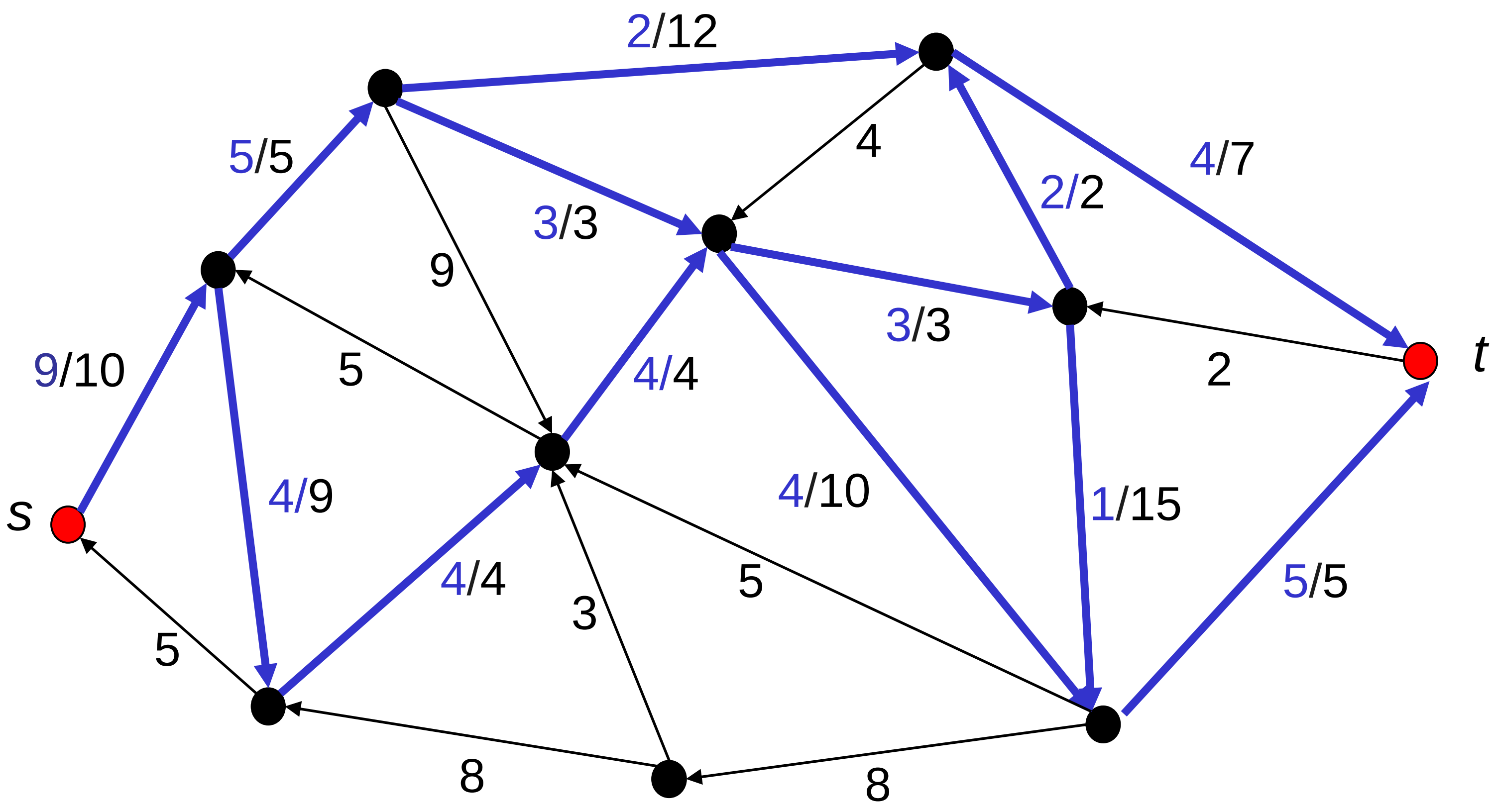
Feasible (s,t)-flow: $f: E \to \mathbb{R}_0^+$

a) $0 \leq f(e) \leq c(e) \quad \forall e \in E$ Capacity constraints

b) $\sum_{e \in in(v)} f(e) = \sum_{e \in out(v)} f(e) \quad \forall v \in V \setminus \{s,t\}$ Flow conservation

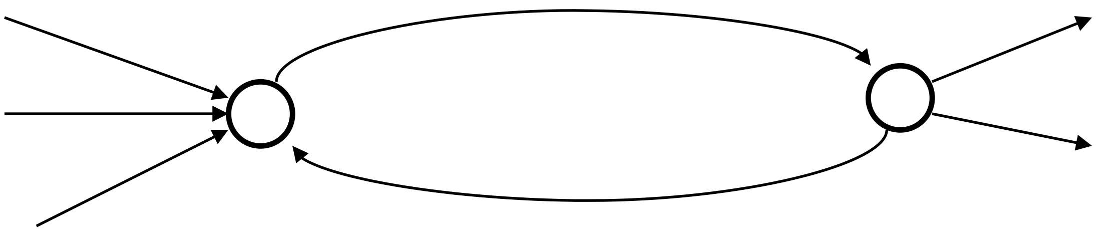
$$in(v) = \{\text{edges into } v\} \qquad out(v) = \{\text{edges out of } v\}$$

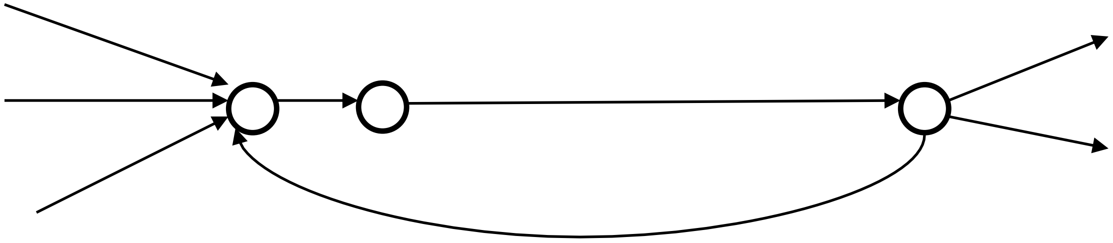
Example



Forward / backward edges

W.l.o.g. graph *G* has no pair of forward / backward edges.





Value of a flow

Let *f* be a feasible flow. Then its value is:

$$V(f) = \sum_{e \in out(s)} f(e) - \sum_{e \in in(s)} f(e)$$

The max-flow problem:

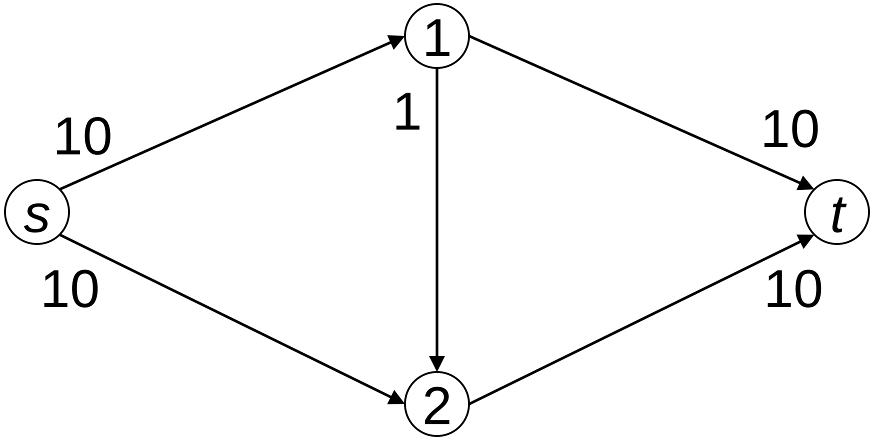
Compute a feasible flow of maximum value.

2. Cuts

Definition: An (s,t)-cut is a partition S,T of V, i.e. $V = S \cup T$, $S \cap T = \emptyset$, such that $s \in S$, $t \in T$.

Capacity of a cut:

$$C(S,T) = \sum_{e \in E \cap (S \times T)} c(e)$$



Flows and cuts

Lemma 1: Let *f* be a feasible flow and (*S*,*T*) be an (*s*,*t*)-cut. It holds that

$$V(f) \leq C(S,T).$$

Proof:

$$\begin{aligned} V(f) &= \sum_{e \in out(s)} f(e) - \sum_{e \in in(s)} f(e) \\ &= \sum_{v \in S} \left(\sum_{e \in out(v)} f(e) - \sum_{e \in in(v)} f(e) \right) \\ &= \sum_{e \in E \cap (S \times T)} f(e) - \sum_{e \in \in E \cap (T \times S)} f(e) \\ &\leq \sum_{e \in E \cap (S \times T)} c(e) \\ &= C(S,T) \end{aligned}$$

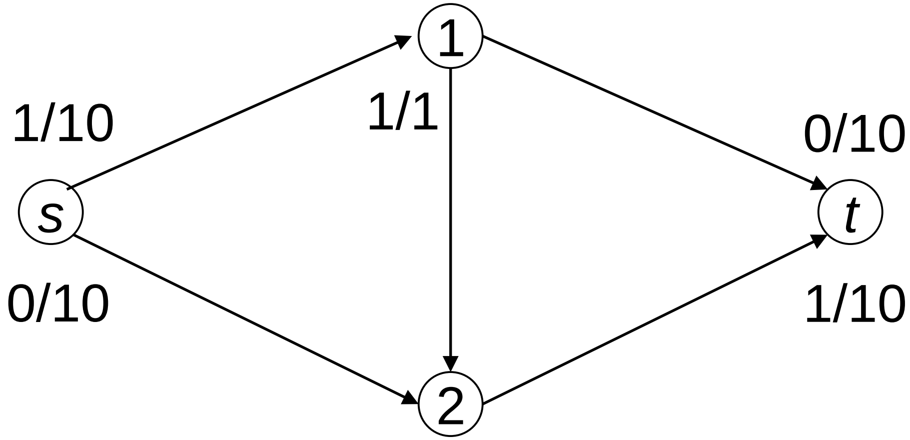
Flows and cuts

Theorem 1: Let f be a flow of maximum value and (S,T) be an (s,t)-cut of minimum capacity. It holds that

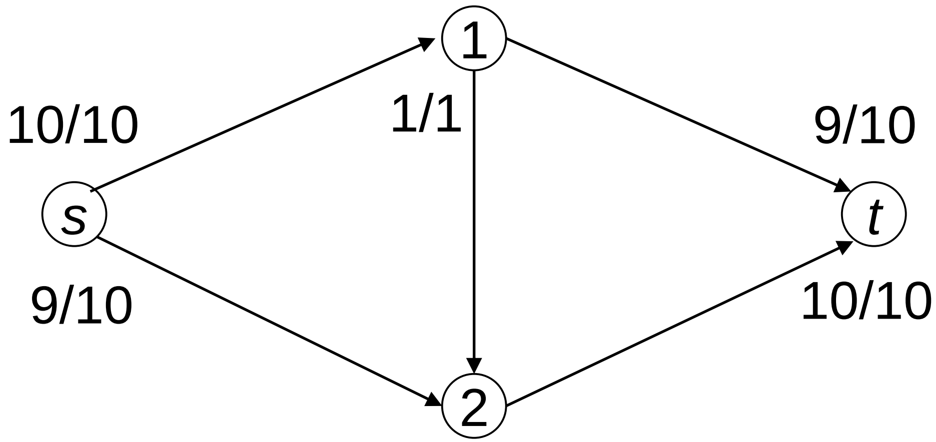
$$V(f) = C(S,T).$$

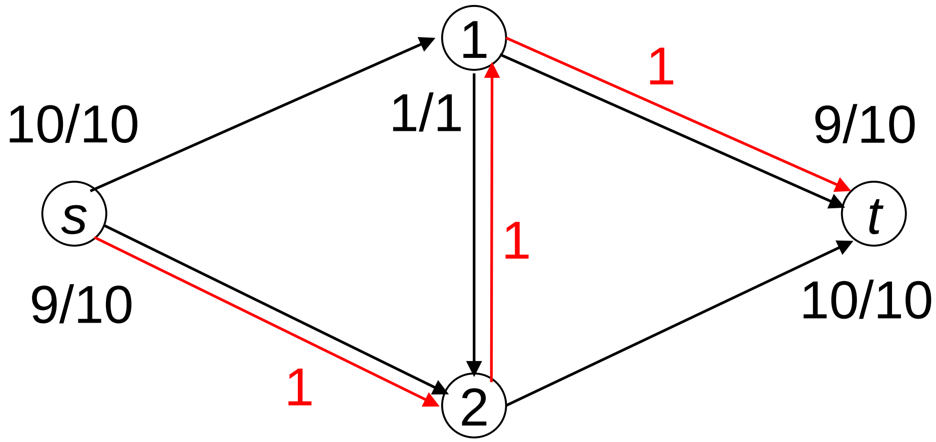
3. Algorithmic idea

Augmenting paths: Find paths along which the flow can be increased.

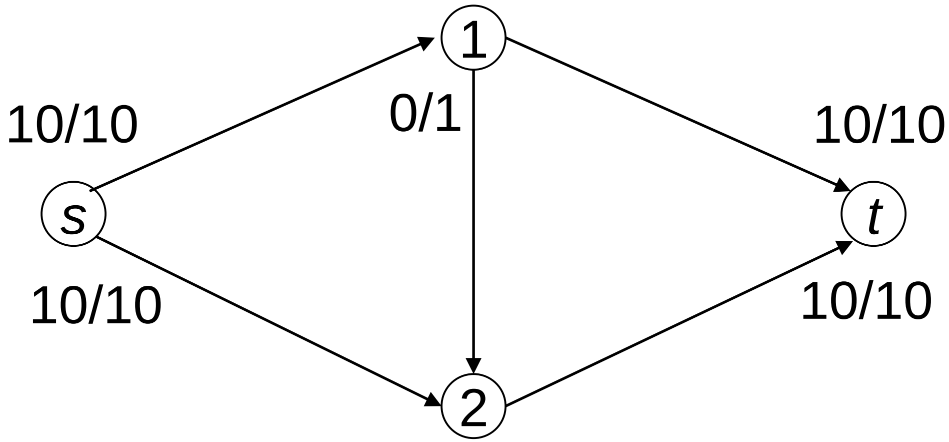


Augmenting paths





Augmenting paths



4. Residual network

Residual network *RN* for a given feasible flow *f*:

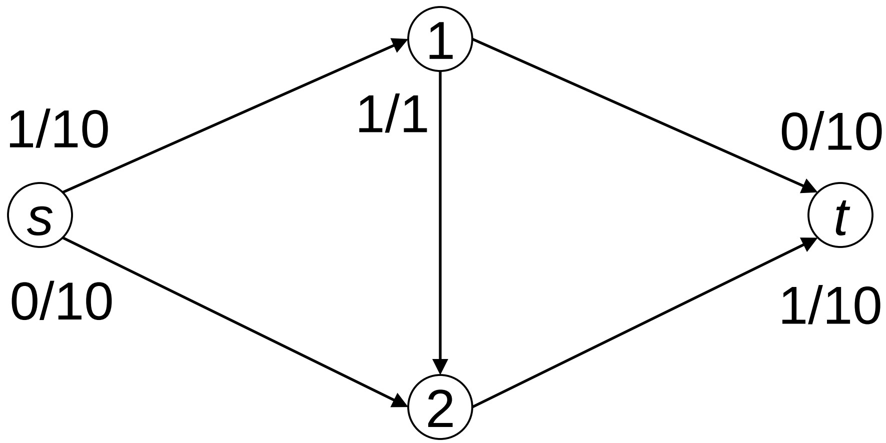
$$E_1 = \{ (v,w) : (v,w) = e \in E \text{ and } f(e) < c(e)\}$$
$$E_2 = \{ (w,v) : (v,w) = e \in E \text{ and } f(e) > 0\}$$

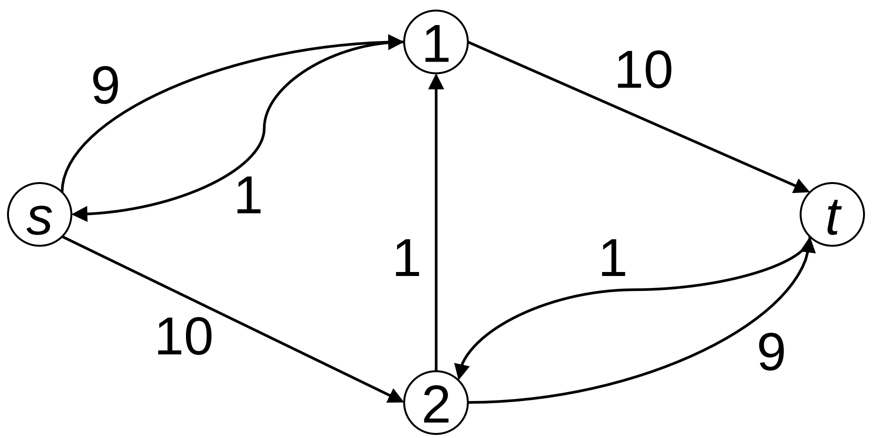
For $e = (v,w) \in E$ use e_1 for $(v,w) \in E_1$ (if it exists)
e_2 for $(w,v) \in E_2$ (if it exists)

$$\bar{c} : E_1 \cup E_2 \to R^+ \qquad \bar{c}(e_1) = c(e) - f(e) \quad \text{for } e_1 \in E_1$$
$$\bar{c}(e_2) = f(e) \quad \text{for } e_2 \in E_2$$

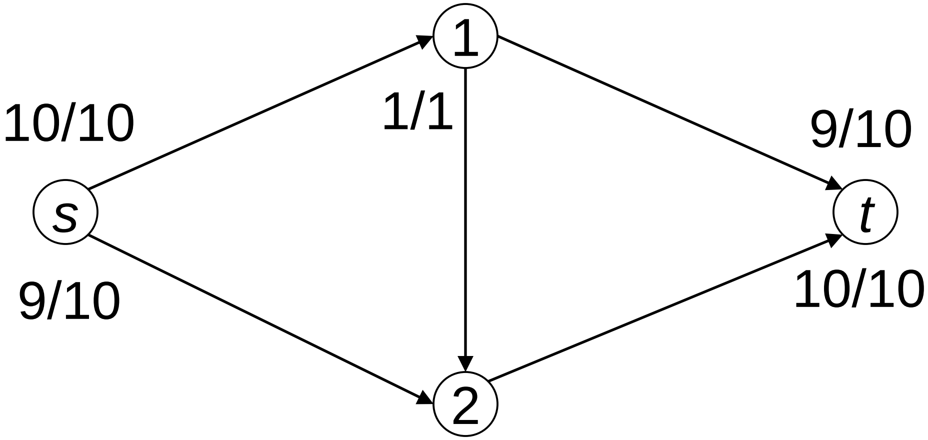
$RN = (V,\ E_1 \cup E_2,\ \bar{c})$

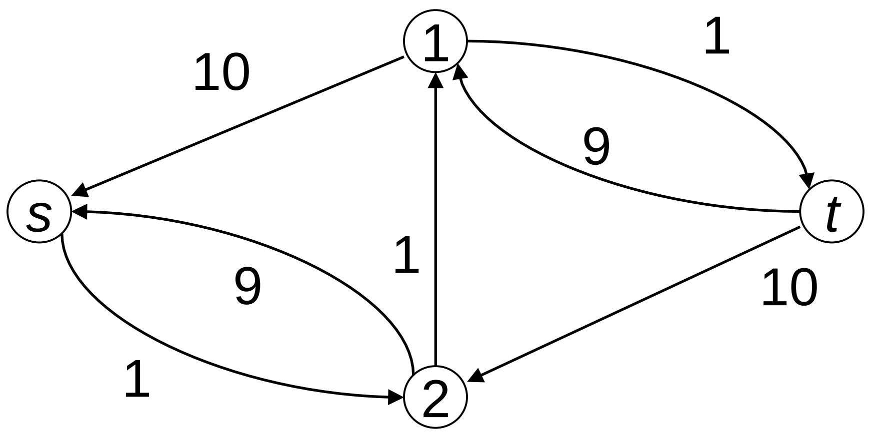
Example





Example





Level network

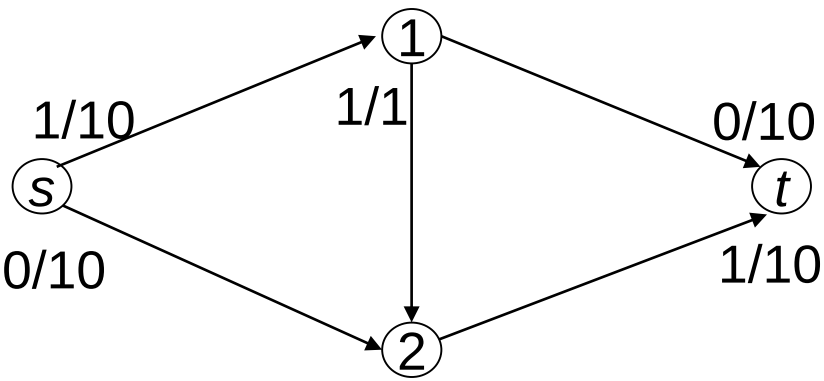
$$V_0 = \{s\}$$

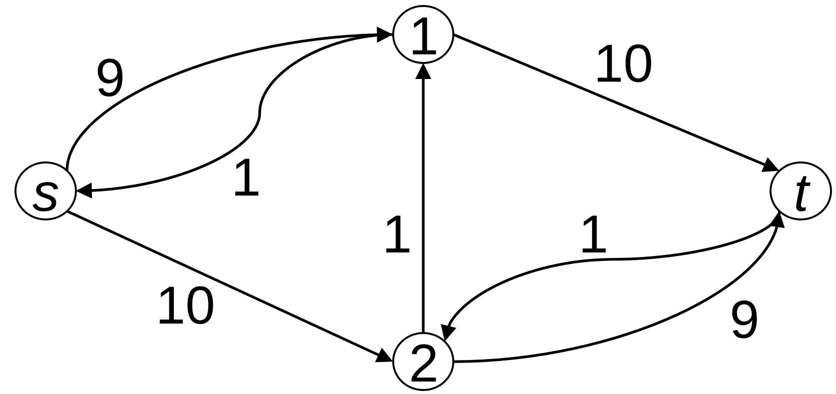
$$V_{i+1} = \{w \in V - (V_0 \cup \ldots \cup V_i); \quad \exists v \in V_i : (v, w) \in E_1 \cup E_2\} \quad \text{for } i \geq 1$$

$$\bar{V} = \bigcup_{i \geq 0} V_i$$

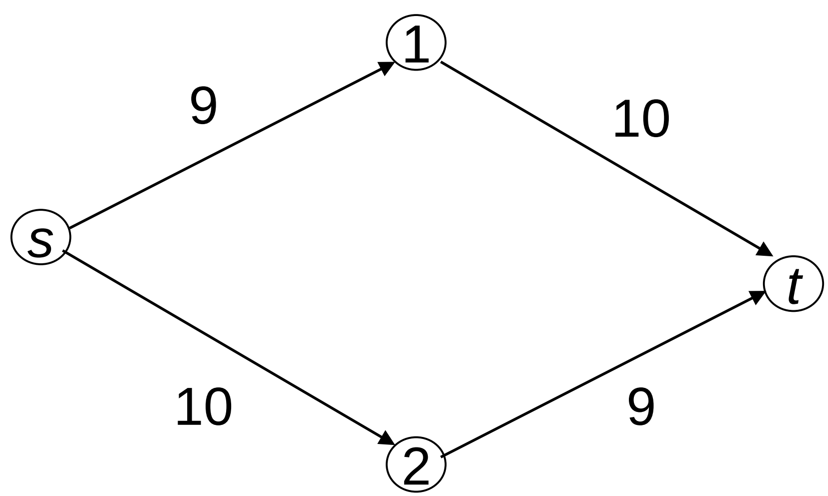
$$LN = \left(\bar{V}, \left(E_1 \cup E_2 \right) \cap \bigcup_{i \geq 0} \left(V_i \times V_{i+1} \right), \bar{c} \right)$$

Example

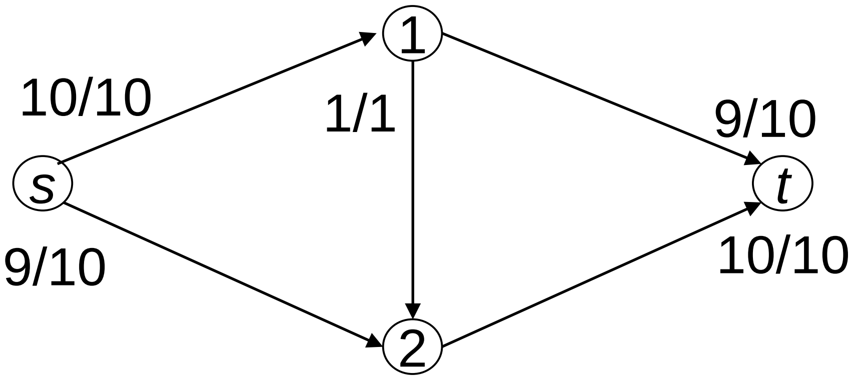


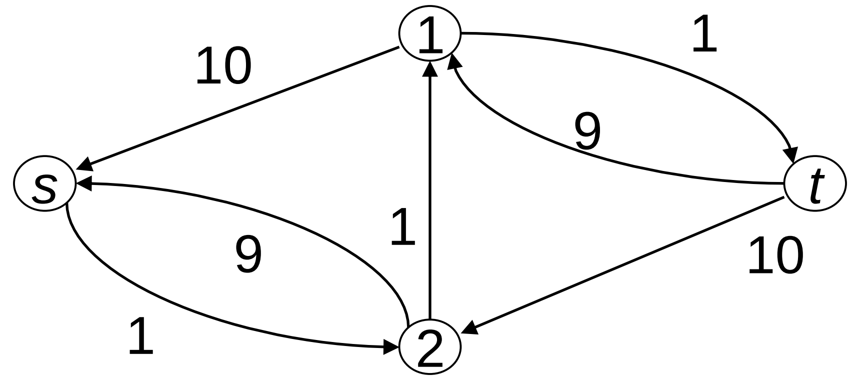


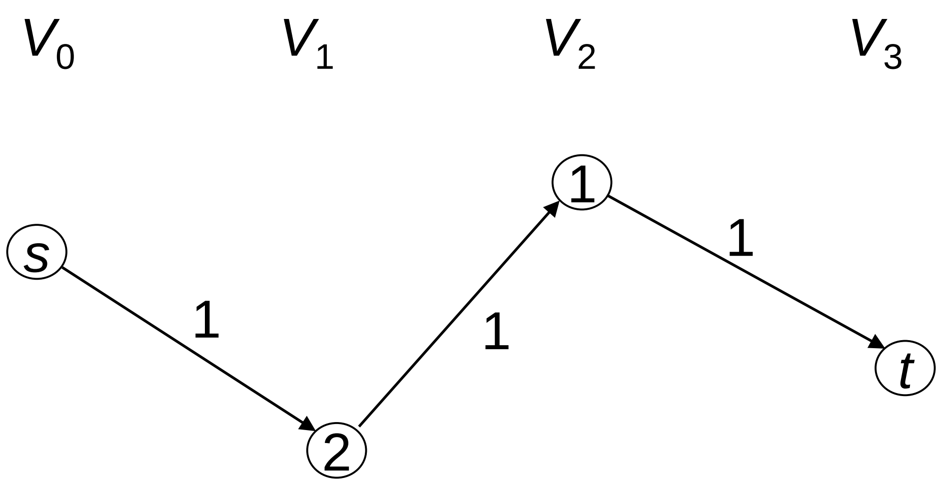
V_0 V_1 V_2



Example







Maximum flows

Lemma 2: Sei f be a feasible flow in N and let $LN = (\bar{V}, \bar{E}, \bar{c})$ be the level network for f.

a) f is a maximum flow if and only if $t \notin \bar{V}$.

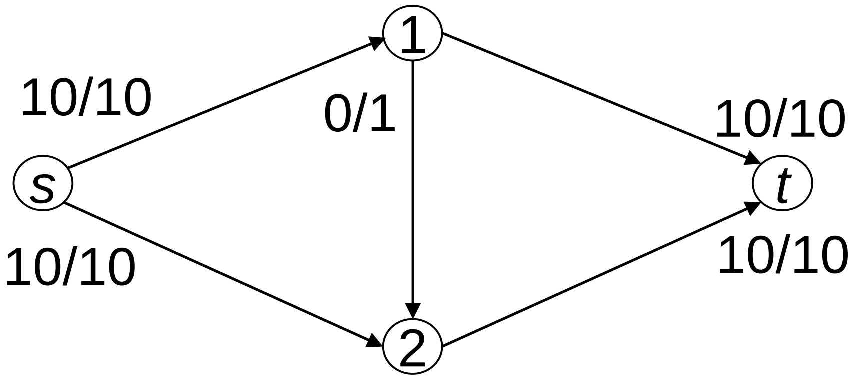
b) Let $\bar{f}$ be a feasible flow in LN. Then $f': E \rightarrow \mathbb{R}$ with

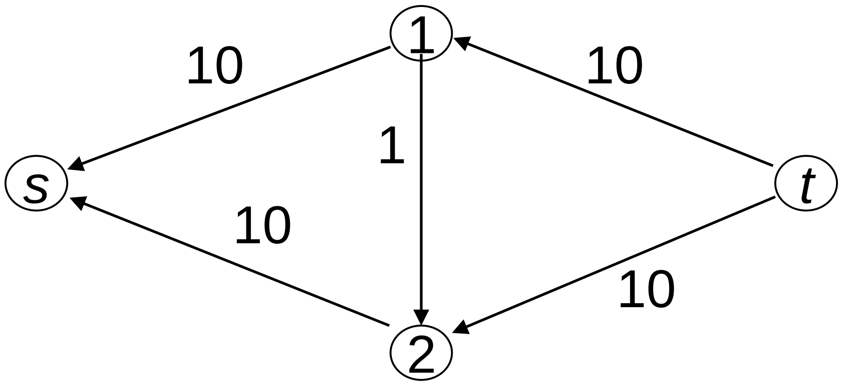
$$f'(e) = f(e) + \bar{f}(e_1) - \bar{f}(e_2)$$

is a feasible flow in N with $V(f') = V(f) + V(\bar{f})$.

Define $\bar{f}(e_i) = 0$ for $e_i \notin \bar{E}$.

Example





Proof, part b)

Proof: b) Capacity constraints. For any $e \in E$, it holds that:

$$
\begin{aligned}
0 &\le f(e) - \bar{f}(e_2) \\
&\le f(e) + \bar{f}(e_1) - \bar{f}(e_2) \\
&= f'(e) \\
&= f(e) + \bar{f}(e_1) - \bar{f}(e_2) \\
&\le f(e) + \bar{f}(e_1) \\
&\le c(e)
\end{aligned}
$$

The first inequality holds because $\bar{f}(e_2) \le \bar{c}(e_2) = f(e)$.

The last inequality follows because $\bar{f}(e_1) \le \bar{c}(e_1) = c(e) - f(e)$.

Proof, part b)

For every $v \in V$, it holds that:

$$\sum_{e \in out(v)} f'(e) - \sum_{e \in in(v)} f'(e)$$

$$= \sum_{e \in out(v)} f(e) - \sum_{e \in in(v)} f(e) + \left(\sum_{e \in out(v)} \overline{f}(e_1) + \sum_{e \in in(v)} \overline{f}(e_2) \right)$$

$$- \left(\sum_{e \in in(v)} \overline{f}(e_1) + \sum_{e \in out(v)} \overline{f}(e_2) \right)$$

Flow conservation: For every $v \in V \backslash \{s,t\}$, the last expression is equal to 0.

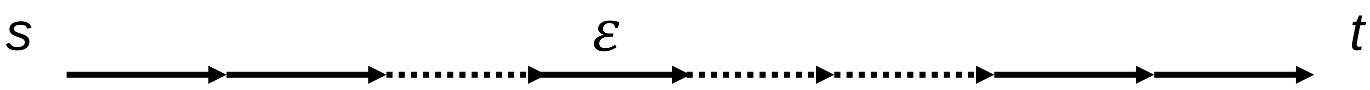
Value: For $v=s$, we obtain $V(f') = V(f) + V(\overline{f})$.

Proof, part a)

a) $"\Rightarrow"$

Let $t \in \overline{V}$.

Then there exists a path *P* from *s* to *t* in *LN*.



ε := min. capacity of any edge in *P*

$$\overline{f}(e) := \begin{cases} \varepsilon & e \text{ in } P \\ 0 & e \text{ not in } P \end{cases}$$

Adding $\overline{f}$ to *f*, as specified in part b) of the lemma, yields a flow of higher value. Hence *f* is not a maximum flow.

Proof, part a)

$$"\Leftarrow"$$

Let $S = \overline{V}, \quad T = V - S$

It holds that $s \in S, \quad t \in T$. Hence (S,T) is an $(s,t)-$cut.

$(E_1 \cup E_2) \cap (S \times T) = \emptyset$

$$f(e) = c(e) \quad \text{for } e \in S \times T$$
$$f(e) = 0 \quad \text{for } e \in T \times S$$

$$V(f) = \sum_{e \in E \cap (S \times T)} f(e) - \sum_{e \in E \cap (T \times S)} f(e) = C(S,T)$$

The first equation above was shown in the proof of Lemma 1.
Since $V(g) \leq C(S,T)$, for every feasible flow g, flow f is a maximum flow.

Max-flow min-cut theorem

Theorem 1: Let $N = (V, E, c)$ be a network and $s, t \in V$.

V_{max} = maximum value of a feasible (s,t)-flow

C_{min} = minimum capacity of an (s,t)-cut

$$V_{max} = C_{min}$$

Proof: By Lemma 1 we have $V_{max} \leq C_{min}$.

Let f be a flow with $V(f) = V_{max}$ and let $LN = (\bar{V}, \bar{E}, \bar{c})$ be the level network for f.

Set $S = \bar{V}$ and $T = V - S$. In the proof of Lemma 2 we showed

$$V(f) = \sum_{e \in E \cap (S \times T)} f(e) - \sum_{e \in E \cap (T \times S)} f(e) = C(S,T).$$

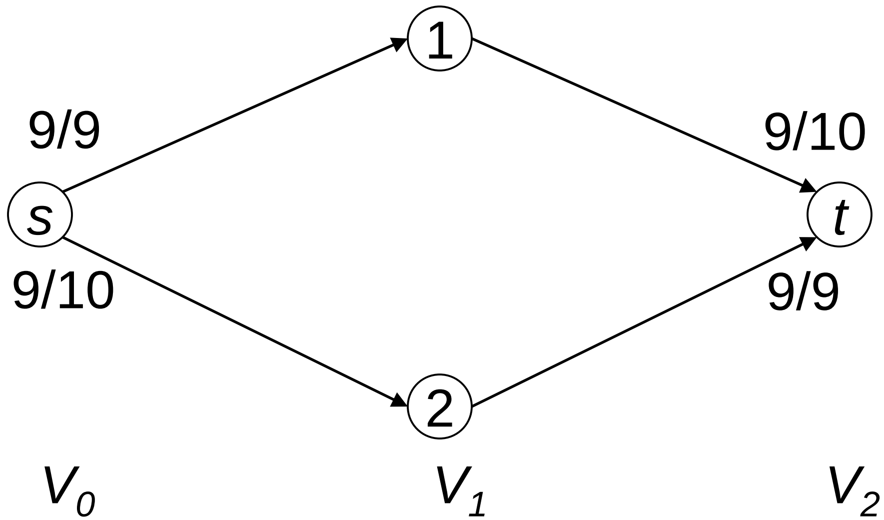
Using the fact that $V_{max} \leq C_{min}$, it follows that (S,T) is an (s,t)-cut of minimum capacity.

5. Blocking flows

Definition: A feasible flow $\overline{f}$ in a level network *LN* is a blocking flow if on every path

$$s = v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \xrightarrow{e_3} \quad \ldots \quad \xrightarrow{e_k} v_k = t$$

from *s* to *t* at least one edge is *saturated,* i.e. $\overline{f}(e_i) = \overline{c}(e_i)$ for at least one *i*.



Algorithm

1. $f(e) := 0$ for all $e \in E$;
2. Construct the level network $LN = (\bar{V}, \bar{E}, \bar{c})$ for f;
3. **while** $t \in \bar{V}$ **do**
4. Find a blocking flow $\bar{f}$ in LN;
5. Update f using $\bar{f}$ as specified in Lemma 2b);
6. Construct the level network LN for f;
7. **endwhile**;

How do we find a blocking flow?

How many iterations?

6. Depth of a level network

Definition: The depth of a level network is the value k with $t \in V_k$.

Lemma 3: Let k_i be the depth of the level network in the i-th iteration.
It holds that $k_i > k_{i-1}$, for $i \geq 2$.

Proof: Level network in the i-th iteration: LN_i

There exists a path P from s to t of length k_i.

$$s = v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \xrightarrow{e_3} \;\ldots\ldots\; \xrightarrow{e_{k_{i-1}}} v_{k_{i-1}} \xrightarrow{e_{k_i}} v_{k_i} = t$$

d_j = level number of v_j in LN_{i-1}, $0 \leq j \leq k_i$

$d_j = \infty$ if v_j is no vertex in LN_{i-1}

Depth of a level network

Claim:

For every $i \geq 2$ it holds that:

a) If there exists an edge from $v_{j\text{-}1}$ to v_j in $LN_{i\text{-}1}$, then $d_j = d_{j-1} + 1$.

b) If there exists no edge from $v_{j\text{-}1}$ to v_j in $LN_{i\text{-}1}$, then $d_j \leq d_{j-1}$.

c) $k_{i-1} < k_i$

Proof:

a) Obvious.

Part b)

b) Assumption: $d_j \geq d_{j-1} + 1$

f_{i-1} yields LN_{i-1} f_i yields LN_i

Case 1. $(v_{j-1},v_j) \in E$: Since $d_j \geq d_{j-1} + 1$, vertex v_j is not contained in levels numbered 0 to d_{j-1} in LN_{i-1}. If there is no edge from v_{j-1} to v_j in LN_{i-1}, then (v_{j-1},v_j) is not in the residual network for f_{i-1}. Thus $f_{i-1}(v_{j-1},v_j) = c(v_{j-1}, v_j)$ and (v_j,v_{j-1}) is in the residual network for f_{i-1}. Since (v_{j-1},v_j) is an edge in RN_i, we have $f_i(v_{j-1},v_j) < c(v_{j-1}, v_j) = f_{i-1}(v_{j-1},v_j)$ and flow along (v_{j-1},v_j) was reduced. It follows that $(v_j,v_{j-1}) \in E_{i-1}$.

Case 2. $(v_j, v_{j-1}) \in E$: We have $f_{i-1}(v_j,v_{j-1}) = 0$ since otherwise (v_{j-1},v_j) would be in the residual network for f_{i-1} and would be included in LN_{i-1}, given that $d_j \geq d_{j-1} + 1$. Moreover, $f_i(v_j,v_{j-1}) > 0$ because (v_{j-1},v_j) is in LN_i. Hence flow was increased along (v_j,v_{j-1}) and $(v_j,v_{j-1}) \in E_{i-1}$.

In any case $(v_j, v_{j-1}) \in \overline{E}_{i-1}$. Therefore $d_{j-1} = d_j + 1$ and $d_j = d_{j-1} - 1 < d_{j-1}$.

Part c)

c) Since $v_0 = s$ and $d_0 = 0$, parts a) and b) imply $d_j \leq j$, for $1 \leq j \leq k_i$.

In particular $k_{i-1} = d_{k_i} \leq k_i$.

We next argue that there exists in edge (v_{j-1},v_j) on the path P in LN_i that does not exist in LN_{i-1}. Suppose on the contrary that all edges of P exist in LN_{i-1}. The computed blocking flow $\bar{f}_{i-1}$ saturates at least one edge (v_{j-1},v_j) of P in LN_{i-1}.

If $(v_{j-1},v_j) \in E$, then $\bar{f}_{i-1}(v_{j-1},v_j) = \bar{c}_{i-1}(v_{j-1},v_j) = c(v_{j-1},v_j) - f_{i-1}(v_{j-1},v_j)$. Note that the reverse edge (v_j,v_{j-1}) is not contained in LN_{i-1}. It follows that $f_i(v_{j-1},v_j) = f_{i-1}(v_{j-1},v_j) + \bar{f}_{i-1}(v_{j-1},v_j) = c(v_{j-1},v_j)$ and (v_{j-1},v_j) is not contained in LN_i.

If $(v_j,v_{j-1}) \in E$, then $\bar{f}_{i-1}(v_{j-1},v_j) = \bar{c}_{i-1}(v_{j-1},v_j) = f_{i-1}(v_j,v_{j-1})$. Again (v_j,v_{j-1}) is not contained in LN_{i-1}. It follows that $f_i(v_j,v_{j-1}) = f_{i-1}(v_j,v_{j-1}) - \bar{f}_{i-1}(v_{j-1},v_j) = 0$ and (v_{j-1},v_j) is not contained in LN_i.

In both cases we obtain a contradiction.

Part c)

Consider path P in LN_i.

$$s = v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \xrightarrow{e_3} \;\; ..... \;\; \xrightarrow{e_{k_{i-1}}} v_{k_{i-1}} \xrightarrow{e_{k_i}} v_{k_i} = t$$

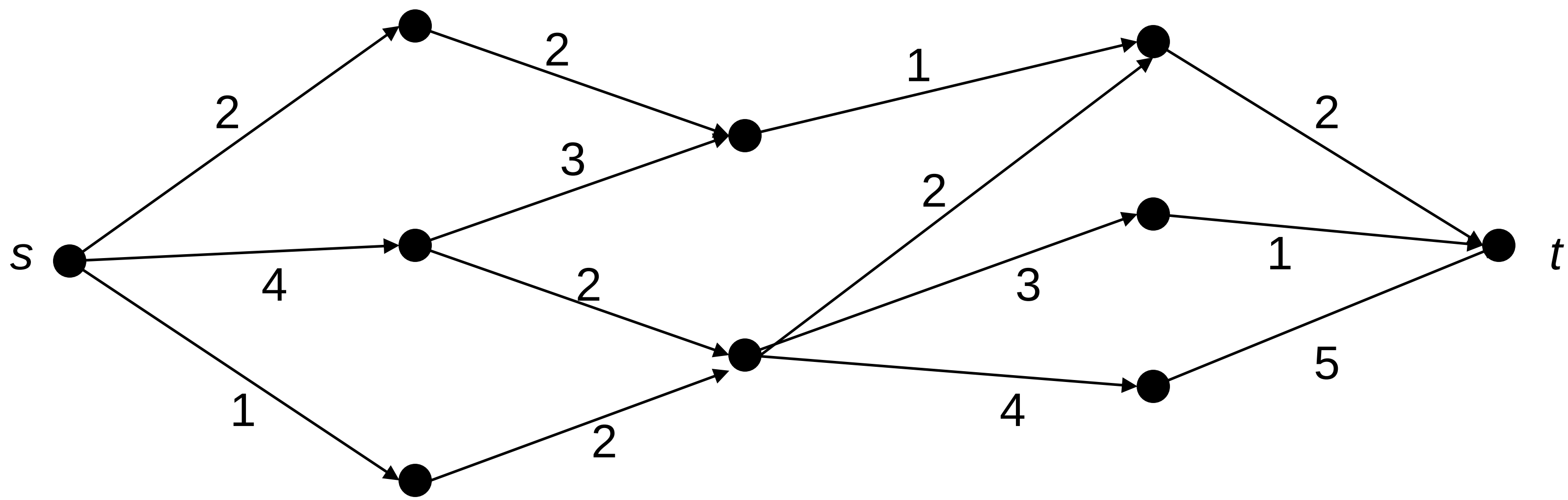
Let (v_{j-1}, v_j) be the edge not contained in LN_{i-1}. Part a) and b) imply $d_{j-1} \leq j$-1. Part b) ensures $d_j \leq j$-1. Again, by parts a) and b), along each of the remaining k_i - j edges of P the level number can increase by at most 1.

We conclude $k_{i-1} = d_{k_i} \leq j - 1 + k_i - j < k_i$.

Number of iterations

Corollary: The number of iterations is $\leq n$.

7. Blocking flows: DFS algorithm



Starting at *s*, at any vertex always choose the first outgoing edge until
a) *t* is reached or b) a dead end *v* (no outgoing edges) is reached.

a) Determine the minimum capacity ε along the path. Increase the flow by ε, reduce the capacity by ε and delete saturated edges.

b) Go back one vertex, delete *v* and its incoming edges.

Analysis

Let $n=|V|$ and $m=|E|$.

Theorem 2: A blocking flow can be computed in time $O(nm)$.

Proof: k = depth of the level network

Construction of a path requires time
$O(k +$ # traversed edges ending in a dead end).

At most m paths are constructed because on each path at least one edge gets saturated. Every edge, over all path constructions, ends only once in a dead end.

Total time: $O(km + m) = O(nm)$

Improved algorithm

Work with the level network. Maintain a working copy that is used to construct a blocking flow. A second copy keeps track of the flow constructed so far.

Potential of a vertex *v*

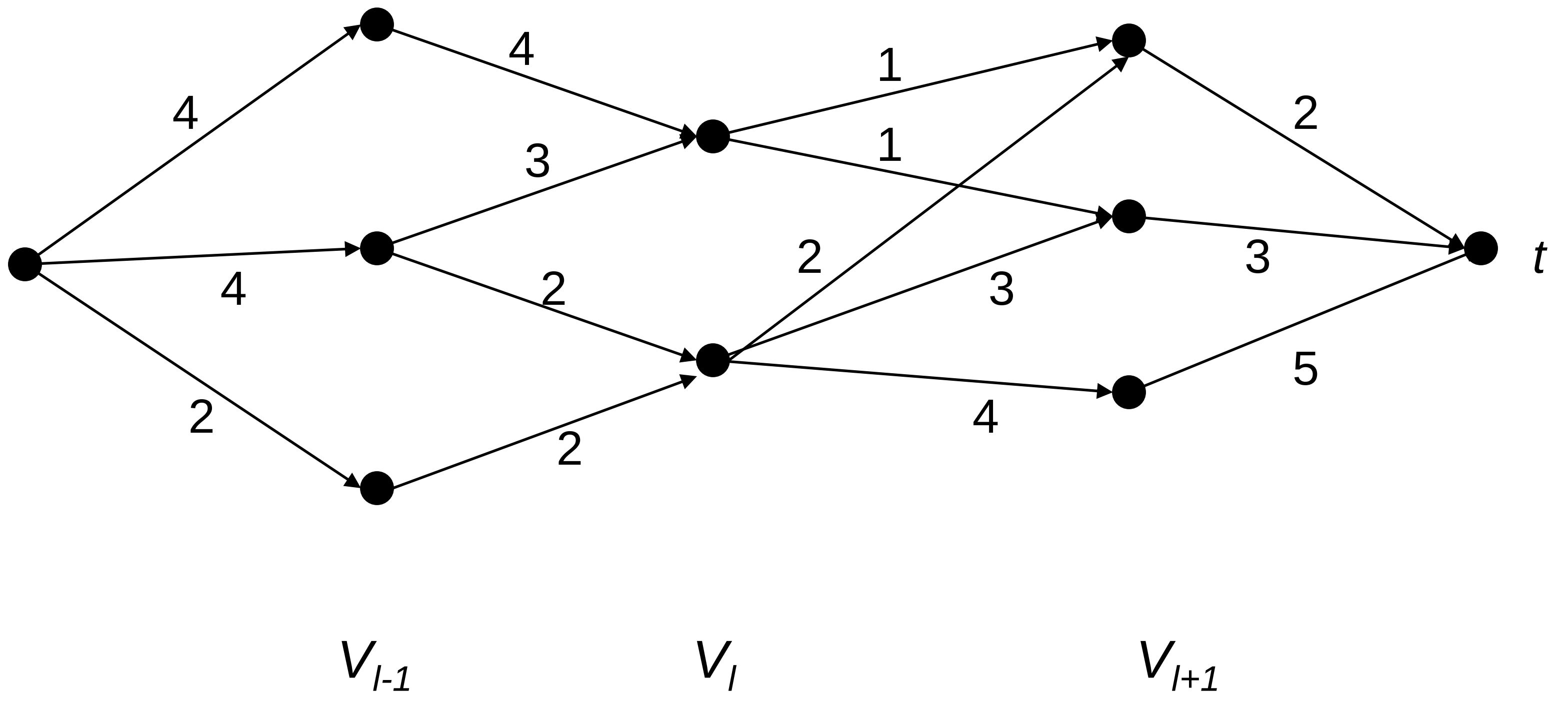
$$P(v) = \min\left\{ \sum_{e \in out(v)} \overline{c}(e), \sum_{e \in in(v)} \overline{c}(e) \right\}$$

$P^* = \min\{P(v) : v \in \overline{V}\}$

Improved algorithm

Choose *v* with *P*(*v*) = *P**.

Push *P** flow units from *v* to higher levels.



Improved algorithm

Level V_h: $S_h \subseteq V_h$, set containing P^* extra flow units

$$P^* = \sum_{x \in S_h} S[x] \qquad S[x] = \text{supply at vertex } x$$

Pull P^* flow units into v from lower levels.

Flow increases by P^* units.

Simplify the network by deleting saturated edges and vertices with indegree or outdegree equal to 0 (at least one vertex is deleted).

Pushing flow

Algorithm *push*(x,S,h);

\\ x is vertex in level V_h and has a supply of S extra flow units to be pushed to vertices in level V_{h+1}.

1. **while** S >0 **do**
2. Let e = (x,y) be the first outgoing edge at x;
3. δ := min{S, $\bar{c}(e)$};
4. Increase the flow along e by δ, reduce $\bar{c}(e)$ by δ,
   add y to S_{h+1} (in case y is not yet element), increase $S[y]$ by δ;
5. S := S - δ;
6. **if** $\bar{c}(e)$ = 0 **then** delete e from the network **endif**;
7. **endwhile;**
8. Delete x from S_h and set $S[x]$:=0;
9. **if** $out(x) = \emptyset$ and $x \neq t$ **then**
10. Add x to the set *del*;
11. **endif**;

Algorithm computing a blocking flow

1. **for all** $x \in V$ **do** $S[x] := 0$; **endfor;**
2. **for all** l, $0 \le l \le k$, **do** $S_l := \emptyset$; **endfor;**
3. $del \leftarrow \emptyset$;
4. **while** LN is not empty **do**
5. Compute $P[v]$ for all $v \in V$ and $P^* = \min \{P[v]; v \in V\}$;
   Let $v \in V_l$ be a vertex with $P^* = P[v]$;
6. $S[v]:=P^*$; $S_l:=\{v\}$;
7. **for** $h:= l$ **to** $k-1$ **do**
8. **for all** $x \in S_h$ **do** $push(x,S[x], h)$; **endfor;**
9. **endfor;**
10. $S[v]:=P^*$; $S_l:=\{v\}$
11. **for** $h:= l$ **downto** 1 **do**
12. **for all** $x \in S_h$ **do** $pull(x,S[x],h)$ **endfor;**
13. **endfor;**
14. $simplify(del)$;
15. **endwhile;**

Result

Theorem 3: A blocking flow in a level network can be compute in time $O(n^2)$.

Proof: 1-3: $O(n)$

Loop 4-15: Executed $O(n)$ times. Each execution takes $O(n)$ if we ignore *push*, *pull*, *simplify*.

All executions of *push* / *pull* take time $O(n^2 + e)$: If a push/pull operation at x (line 4) does not saturate an outgoing/incoming edge e, i.e. $\bar{c}(e)$ remains positive, then the operation terminates the current call of *push* / *pull*.

All executions of *simplify* take time $O(n + m)$.

Main result

Theorem 4: A maximum flow can be computed in $O(n^3)$ time.

Proof: There are at most n iterations. In each one, a level network and a blocking flow can be computed in time $O(n^2)$.

8. d-bounded networks

Definition: Let $d \in \mathbb{N}$. $N = (V,E,c)$ is d-bounded if $c(e) \in \{1,2,...,d\}$ for all $e \in E$.

1-bounded networks are called (0,1)-networks.

Application of our flow algorithms to d-bounded networks:

→ all computed flows are integral, i.e. $f(e) \in \mathbb{N}_0$

→ the maximum flow is integral

d-bounded networks

Theorem 5: A blocking flow in a d-bounded network can be computed in time $O(dm)$. For d = 1 we obtain $O(m)$ time.

Proof: DFS algorithm

Time for the construction of a path:

O(# edges on s-t-path + # traversed edges ending in a dead end)

Each edge is contained in at most d paths.

Maximum flows in residual networks

Lemma 4: Let N be a network and V_{max} be the value of a maximum (s,t)-flow. Let RN be the residual network for a flow f and $\overline{V}_{max}$ be the value of a maximum (s,t)-flow in RN. It holds that

$$V_{max} = \overline{V}_{max} + V(f).$$

Proof: Let (S,T) be an (s,t)-cut.

$C(S,T)$: capacity of (S,T) in N

$\overline{C}(S,T)$: capacity of (S,T) in RN

$$\overline{C}(S,T) = \sum_{v \in S, w \in T} \overline{c}(v,w) = \sum_{v \in S, w \in T} (c(v,w) - f(v,w) + f(w,v))$$

$$= C(S,T) - \left(\sum_{v \in S, w \in T} f(v,w) - \sum_{v \in S, w \in T} f(w,v) \right)$$

$$= C(S,T) - V(f)$$

Maximum flows in residual networks

We obtain

$$\overline{C}_{\min} = C_{\min} - V(f),$$

where $C_{\min}$ and $\overline{C}_{\min}$ denote the minimum capacities of (s,t)-cuts in N and RN, respectively.

Using the max-flow min-cut theorem we conclude

$$\overline{V}_{\max} = V_{\max} - V(f).$$

9. Simple networks

Definition: A network $N = (V,E,c)$ is simple, if $indeg(v) = 1$ or $outdeg(v) = 1$, for all $v \in V \setminus \{s,t\}$.

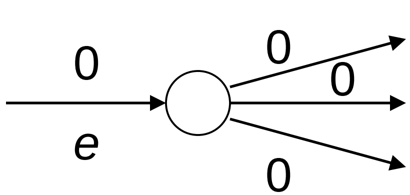
Theorem 6: Let $N = (V,E,c)$ be a simple (0,1)-network. Then a maximum flow can be computed in time $O(n^{1/2}m)$.

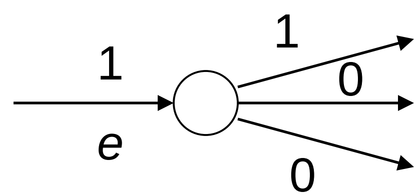
Residual networks of simple networks

Claim: Let N be a simple (0,1)-network and f be an integral flow in N. Then RN is a simple (0,1)-network.

Proof: Sei $v \in V \setminus \{s,t\}$ and $indeg(v) = 1$ ($outdeg(v) = 1$ is analogous).

If $f(e) = 0$ for $e \in in(v)$, then $f(e') = 0$, for all $e' \in out(v)$, and v has indegree 1 in RN.





If $f(e) = 1$ for $e \in in(v)$, then $f(e') = 1$ for exactly one $e' \in out(v)$ and v has indegree 1 in RN.

Obviously, the edge capacities in RN are either 0 or 1.

Proof of Theorem 6

Consider our maximum flow algorithm. All intermediate flows are integral.

A blocking flow can becomputed in time $O(m)$.

We prove: # iterations = $O(n^{1/2})$.

V_{max} = value of a maximum (s,t)-flow

$V_{max} \leq n^{1/2}$: ok

We study the case that $V_{max} > n^{1/2}$.

Let iteration l be the one increasing the flow value to $> V_{max} - n^{1/2}$.

We show that the level network in iteration l has depth $< n^{1/2}+1$.

This implies that before iteration l, at most $n^{1/2}+1$ iterations were executed. After iteration l, at most $n^{1/2}$ iterations can be performed because the flow value increases by at least 1 in each iteration.

Proof of Theorem 6

f : feasible (s,t)-flow immediately before iteration l

RN: residual network for f

By Lemma 4 there exists a flow $\bar{f}$ in *RN* with value

$$\bar{V}_{\max} = V_{\max} - V(f) \geq V_{\max} - \left(V_{\max} - n^{1/2}\right) = n^{1/2}.$$

Since *RN* is a simple (0,1)-network, we may assume that $\bar{f}$ is integral, i.e. $\bar{f}(e) \in \{0,1\}$.

As *RN* is simple, at most one flow unit is routed through each $v \in V \setminus \{s,t\}$.

$\bar{f}$ consists of at least $n^{1/2}$ vertex-disjoint paths from s to t.

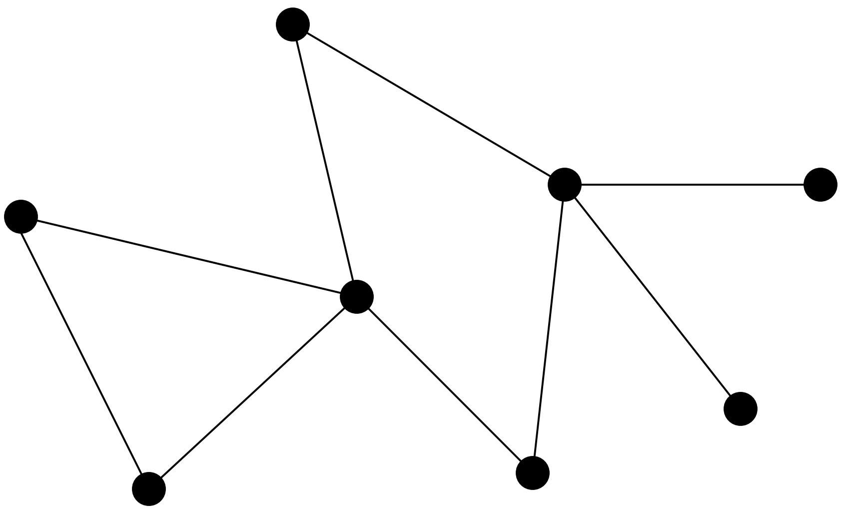
Hence there exists a path with $< n^{1/2}$ intermediate vertices.

10. Matchings in bipartite graphs

$G = (V,E)$ undirected graph

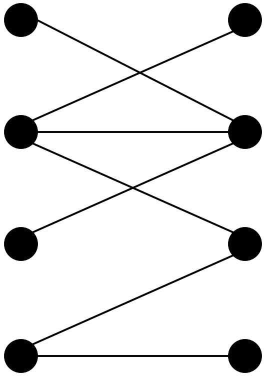
Matching M is an edge set $M \subseteq E$ such that no two edges $e_1, e_2 \in M$, $e_1 \neq e_2$, have a common vertex.

A maximum matching is a matching of maximum cardinality.



Matchings in bipartite graphs

An undirected graph $G = (V,E)$ is bipartite if $V = V_1 \cup V_2$, for $V_1, V_2 \subseteq V$ with $V_1 \cap V_2 = \emptyset$, and $E \subseteq V_1 \times V_2$.



Matchings in bipartite graphs

Theorem 7: Let $G = (V_1 \cup V_2, E)$, $E \subseteq V_1 \times V_2$, be a bipartite graph. Then a maximum matching can be computed in time $O(n^{1/2}m)$.

Proof: Construct a simple network as follows:

(All capacities are equal to 1.)